Evoluția unor imagini pe baza unui algoritm simplu - implementare MPI

25 aprilie 2023

1 Etapa 1

Scopul acestei probleme este scrierea unui program MPI care să implementeze un algoritm de evoluție a unor imagini (format PGM) pe baza unui set minimal de reguli.
Obiective:

- utilizarea unui model de programare de tip master-slave;
- efectuarea unei descompuneri de domeniu (aici o matrice) și schimburi de date în regiuni de tip "halo".

2 Topologia carteziană

Veți aborda problema în etape succesive: într-o primă etapă obiectivele urmărite vor fi:

1. Crearea unui model master-worker – procesul master scrie fișierul de date final (o imagine în format PGM), toate procesele, inclusiv procesul master, dezvoltă algoritmul de definire a valorii pixelilor matricii.
2. Partiționarea unei matrici între procese.
3. Generarea unei topologii carteziene virtuale. Topologia cartezianã, utilizată pe scară largă în programarea MPI, este foarte bine adaptată prelucrărilor de date necesare aici.
4. Fiecare proces va "colora" secțiunea de date locală în alb sau negru, în funcție de poziția sa în structura topologică.
5. Transmiterea de date către procesul master, care va scrie fișierul PGM final (se poate utiliza funcția **pgm_write()**).

Rezultatul va fi o imagine de tip tablă de șah.

3 Algoritmul

Pseudo-codul de luat în considerare este următorul:

- Inițializare MPI
- Pasul 1
  - determină numărul de procese lansate în execuție (funcţii MPI necesare **MPI_Comm_size()**, **MPI_Comm_rank()**)
  - determină nx = număr de procese pe direcția Ox în topologia virtuală (Fig. 1), ny = număr de procese pe direcția Oy în topologia virtuală (funcţii MPI necesare **MPI_Dims_create()**)

– creează o structură carteziană (grid) 2D, periodică (funcţii MPI necesare **MPI_Cart_create()**)

- Pasul 2
  - Procesul 0 (master):
    - alocă dinamic un bloc (valori float) de dimensiune $XSIZE \times YSIZE$
    - inițializează toate elementele cu valoarea $0.75 * CONTRAST$ (o culoare gri)
  - Toate procesele:
    - alocă dinamic o matrice de dimensiune $XSIZE/nx \times YSIZE/ny$
    - inițializează toate elementele cu valoarea $0.75 * CONTRAST$

- Pasul 3
  - află coordonatele carteziene (x, y) ale fiecărui proces (funcții MPI necesare **MPI_Cart_coords()**)
  - dacă $(x + y + 1) \mod 2 == 1$: atribuie valoarea 0 (negru) elementelor din matricea locală
  - altfel : atribuie valoarea $CONTRAST$ (alb) elementelor din matricea locală
  - creează tipuri de date derivate pentru transferul datelor către procesul master (funcții MPI necesare **MPI_Type_vector(), MPI_Type_commit()**)

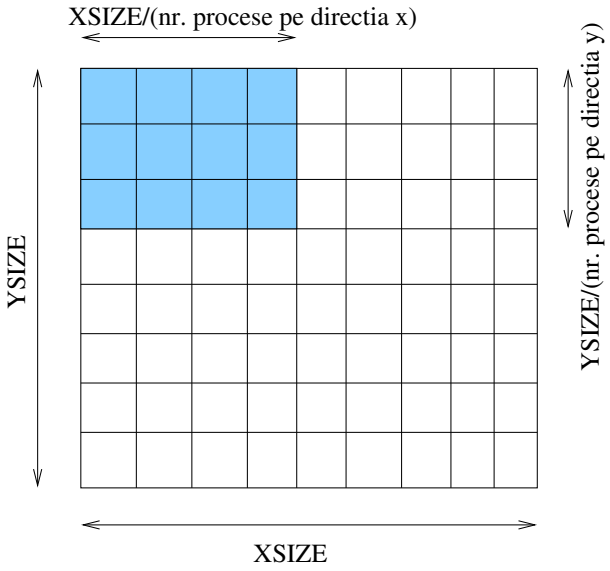

Figura 1: Utilizarea tipurilor de date derivate pentru transfer în blocul menținut de procesul master.

- Pasul 4
  - Procesul 0 (master):
    - copiază datele din blocul local $XSIZE/nx \times YSIZE/ny$ în blocul $XSIZE \times YSIZE$
    - iterează de la rangul 1 la $nproc - 1$
    recepționează datele folosind tipul derivat în zona corespunzatoare a matricii $XSIZE \times YSIZE$ (vezi mai jos)
    - scrie fișierul de date PGM
  - Celelalte procese (worker):
    - transmite datele către procesul master

Datele sunt trimise bloc de procesele worker și recepționate de procesul master ca un tip derivat (creat anterior cu **MPI_Type_vector()**; atenție, și procesul master procesează date!). In matricea (spațiul de adrese) globală de la procesul master, datele vor fi stocate la adresa elementului

$masterdata[dy * coord[1] + coord[0] * dx * YSIZE]$, $dx = XSIZE/nx$, $dy = YSIZE/ny$ (vezi Fig. 1. Verificați!). Functii MPI necesare la pasul 4: **MPI_(I)Send(), MPI_Cart_coords(), MPI_Recv()**.

- Termină mediul MPI

4 Etapa 2 - Schimbul de date între procese în topologia carteziană

Veți modifica programul realizat anterior, pentru a asigura schimbul de date între procese în topologia carteziană, în regiunile de halo. Ca mai înainte, veți începe prin a considera cazul în care numărul de procese pe fiecare direcție în structura topologică virtuală 2D este divizor al $XSIZE$, respectiv $YSIZE$ (generalizarea este simplă, nu este cerută în această aplicație; dacă doriți, o puteți face).

Obiectivele aici vor fi:

- crearea regiunilor de halo pentru fiecare proces:

5 Codul paralel

Structura codului:

```
#include "pgm_IO.h"
#include <mpi.h>

#include "pgm_IO.c"

int main( int argc, char ** argv )
{
  /* ***************** Sectiunea declaratiilor ***************************** */

  int source, my_rank, nproc, tag = 1;
  int NX, // numar de procese distribuite pe directia Ox a structurii carteziene 2
      NY, // numar de procese distribuite pe directia Oy a structurii carteziene 2
      i, j,
      dims[NDIM], // contine distributia optimizata a proceselor pe structura
                  // carteziana 2D
      GSIZE = XSIZE * YSIZE;     // GSIZE este dimensiunea blocului global care
                                 // defineste imaginea

  int reorder = 0, // procesele din reteaua carteziana nu vor fi reordonate
      periods[NDIM]={TRUE,TRUE}; // structura carteziana 2D este periodica

  int dX, // XSIZE/NX
      dY, // XSIZE/NY
      localSize;         // localSize = dX * dY este dimensiunea blocului de date
                  //stocat local pe fiecare procesor

  float *masterdata,  // blocul XSIZE * YSIZE pixeli, stocat doar pe procesul 0
        *data;             // blocul local necesar fiecaruia dintre procese pentru
                    // a-si defini imaginea

  int coords[NDIM];        // sirul coordonatelor procesului in reteaua virtuala
```

```
  // comm_2D este comunicatorul asociat retelei cu NDIM dimensiuni
  MPI_Comm wcomm = MPI_COMM_WORLD, comm_2D;
  MPI_Status status;
  MPI_Datatype blockType;         // tip de date derivat pentru transferul catre
                              // procesul 0 a datelor partiale

  char fname[32];
  strcpy(fname,"chessy_struct.pgm");

  /* ********************************************************************
*/

  /* ********************** PAS 1 ***************************************
   * -initializare MPI
   * -determin NX, numar de procese optim in topologia carteziana pe directia X
   * -determin NY, numar de procese optim in topologia carteziana pe directia Y
   * - creez totpologia carteziana 2D, periodica, fara reordonare
   * ******************************************************************** */
  /* vom determina distributia optima a proceselor pe dimensiuni apeland
   * ulterior MPI_Dims_create() */
  for( i=0; i<NDIM; i++ )
    dims[i] = 0;

 /* aici codul dvs ..... */


  /* ********************** PAS 2 ***************************************
   * Procesul 0:
   * - aloc spatiu RAM pentru XSIZE*YSIZE valori float
   * - initializez spatiul alocat cu 0.75*CONTRAST
   * Celelalte procese:
   * - aloc spatiu RAM pentru deltaX*deltaY intregi
   * - initialilzez spatiul alocat cu 0.75*CONTRAST
   * ******************************************************************** */

   /* aici codul dvs ..... */

  /* ********************** PAS 3 ***************************************
   * - determin coordonatele carteziene ale procesului: (x,y)
   * - daca (x+y+1) mod 2 == 1
   *      atribui 0 (negru) tuturor campurilor din buffer-ul local
   *    altfel
   *      atribui CONTRAST tuturor campurilor din buffer-ul local
   * - creez tipul de date derivat pentru transferul datelor locale catre
   * procesul 0
   * ********************************************************************/

   /* aici codul dvs ..... */


  /* ********************** PAS 4 ***************************************
   * - daca proces cu rang != 0
   *      transmite date brute catre procesul 0
```

```
   * - altfel
   *      itereaza de la procesul 1 la procesul nproc-1
   *      primeste datele in regiunea corecta a buffer-ului, folosind tipul
   *      derivat
   * - scrie datele in fisier
   * - finalizeaza MPI
   * ********************************************************************/

  /* aici codul dvs ..... */

  return 0;
} //end-of-main()
```

6 Etapa 2 - procedura de schimb de date între procese

Obiective:

- crearea unor regiuni de "halo" (regiuni de margine) în fiecare dintre domeniile RAM ale proceselor
- schimbul de date în regiunile de margine

Pe structura programului dezvoltat în prima etapă, veți modifica algoritmul astfel încât, în **pasul 3** veți asigura următoarea funcționalitate:

- definiți și activați două noi tipuri de date derivate (o linie, respectiv o coloană de pixeli din blocul *data*:

  ```
  MPI_Datatype lineType,colType;

  /* linia de pixeli locala in data */
  MPI_Type_vector(1,dY,1,MPI_FLOAT,&lineType);
  MPI_Type_commit(&lineType);

  /* coloana de pixeli locala in data */
  MPI_Type_vector(dX,1,dY,MPI_FLOAT,&colType);
  MPI_Type_commit(&colType);
  ```

- declarați și inițializați variabilele întregi *up, down, left, right* care vor păstra rangurile proceselor vecine celui apelant în topologia virtuală carteziană; astfel, dacă procesul apelant are coordonatele (i, j), atunci *up* este rangul procesului cu coordonatele $(i-1, j)$, *down* este rangul procesului cu coordonatele $(i+1, j)$, etc. Cel mai comod mod de a inițializa aceste variabile este utilizarea funcției **MPI_Cart_shift()**, de exemplu

  ```
  int up, down, left, right;

  /* rangurile vecinilor pe verticala */
  MPI_Cart_shift(comm_2D,0,1,&up,&down);
  ```

  și asemănător pentru cealaltă direcție carteziană.

- trimiteți a doua linie de pixeli din blocul *data* al procesului curent în ultima linie de pixeli ai blocului *data* al procesului cu rangul *up*
- recepționați linia de pixeli de la procesul cu rangul *down* în ultima linie a procesului curent; puteți face asta cu un singur apel al **MPI_Sendrecv()**, funcție creată în acest scop:

```
/* transfer liniile de margine de jos in sus:
   - a doua linie din blocul procesului actual catre ultima linie
     a blocului procesului up
   - receptionez de la procesul down linia transmisa si o stochez
     in ultima linie a procesului actual
*/
MPI_Sendrecv(...);
```

- repetați manevra: penultima linie se trimite către rangul *down* și se recepționează în prima linie linia care vine de la rangul *up*
- repetați cu coloanele de pixeli: penultima coloana din *data* a procesului actual este trimisă către blocul *data* al procesului cu rangul *right* și recepționați în prima coloană vectorul care vine de la procesul cu rangul *left*; a doua coloană din *data* a procesului actual este trimisă către blocul *data* al procesului cu rangul *left* și recepționați în ultima coloană vectorul care vine de la procesul cu rangul *right*. Veți utiliza vectorul *colType* pentru aceste operații.

Programul rezultat ar trebui sa producă o imagine de tipul unei table de șah în care fiecare pătrat este delimitat de un cadru de pixeli de culoare opusă.

7 Finalizare

- Va fi aplicat un algoritm iterativ, definit mai jos, care determină evoluția unei stări inițiale date (o imagine PGM)
- Fiecare proces menține două blocuri locale: blocul curent *data* și blocul "evoluat" *data_next*, definite la fiecare iterație
- Blocurile locale au acum *dX=XSIXE/NX +2* valori (pixeli) pe linie și *dY=YSIXE/NY +2* valori (pixeli) pe coloană; există deci două extra-linii și două extra-coloane pentru regiunile de "halo"
- Tipuri de date derivate:
  - pentru recepție la procesul 0:

    ```
    MPI_Type_vector( deltaX-2, deltaY-2, YSIZE, MPI_FLOAT, &recvBlockType );
    MPI_Type_commit( &recvBlockType );
    ```

  - pentru transmitere, de la procesele de rang nenul (dar toate declară tipurile)

    ```
    MPI_Type_vector( deltaX-2, deltaY-2, dY, MPI_FLOAT, &sendBlockType );
    MPI_Type_commit( &sendBlockType );
    ```

  - coloane și linii, pentru *MPI_Sendrecv()*:

    ```
    MPI_Datatype lineType,colType;

    /* linia de pixeli locala in data */
    MPI_Type_vector(1,dY,1,MPI_FLOAT,&lineType);
    MPI_Type_commit(&lineType);

    /* coloana de pixeli locala in data */
    MPI_Type_vector(dX,1,dY,MPI_FLOAT,&colType);
    MPI_Type_commit(&colType);
    ```

- Care sunt rangurile proceselor vecine in topologia carteziana definita? Pentru definitia regiunilor "halo"...

```
MPI_Cart_shift( comm_2D, 0, 1, &up, &down );
MPI_Cart_shift( comm_2D, 1, 1, &left, &right );
```

- Starea inițială:

```
// conditia initiala - o linie si o coloana de
// pixeli negri aproximativ la mijloc pe Ox si Oy
if( coords[0] == NX/2 )
  {
    for( i=0; i<dX; i++)
      for( j=0; j<dY; j++)
        if( i==dX/2 )
             *(data + i * dY + j) = *(data_next + i * dY + j) = 0;
        else
         *(data + i * dY + j) = *(data_next + i * dY + j) = CONTRAST;
  }
else
if( coords[1] == NY/2 )
  {
    for( i=0; i<dX; i++)
      for( j=0; j<dY; j++)
        if( j==dY/2 )
             *(data + i * dY + j) = *(data_next + i * dY + j) = 0;
        else
         *(data + i * dY + j) = *(data_next + i * dY + j) = CONTRAST;
  }
else
    for( i=0; i<dX; i++)
      for( j=0; j<dY; j++)
           *(data + i * dY + j) = *(data_next + i * dY + j) = CONTRAST;
```

- Se iterează de la 1 la $niter = 20$ (sau altă valoare)
  - Se definesc regiunile "halo" cu cele 4 apeluri *MPI_Sendrecv()*; atenție, regiunile active sunt regiunile de miez (adică zona RAM *data* fără cadrul constiuit din cele două extra-linii și coloane)
  - Reguli de evoluție pentru tiparul inițial ($0 \leq nvec \leq 8$ este numărul vecinilor de ordin I negri ai pixelului curent):
    - Identifică numărul de vecini negri *nvec* ai fiecărui pixel din miezul blocului local *data* conținând starea curentă
    - Aplică regulile de evoluție (se aplică doar miezului!), pixel cu pixel:
      - dacă $nvec < 2$ și pixelul din *data* este negru, pixelul corespunzător din *data_next* devine alb ("nu se poate reproduce")
      - dacă $nvec == 2$ pixelul își păstrează culoarea (se copiază valoarea din *data* în *data_next*)
      - dacă $nvec >= 3$ și $nvec <= 7$ și pixelul din *data* este alb, pixelul corespunzător din *data_next* devine negru ("celula se naște")
      - dacă $nvec > 7$ și pixelul din *data* este negru, pixelul corespunzător din *data_next* devine alb ("suprapopulare")
  - Înainte de a trece la iterația următoare, starea inițială a miezului la iterația următoare este starea evoluată a miezului acestei iterații: se copiază pixelii din *data_next* în *data*
  - La fiecare două iterații:
    - dacă proces cu rang != 0 transmite date brute catre procesul 0

    - altfel
        - iterează de la procesul 1 la procesul $nproc - 1$:
        - primește datele în regiunea corectă a buffer-ului *masterdata*, folosind tipul derivat
        - scrie datele din *masterdata* în fișier
- Eliberează resursele alocate, finalizează MPI *

Succes!